Ajax-sovellusten suunnittelu ja toteutus

Tampereen ammattikorkeakoulu
Viestinnän koulutusohjelman opinnäytetyö
Vuorovaikutteisuuden suunnittelu
Syksy 2008
Sami Harju

OPINNÄYTTEEN TIIVISTELMÄ

Sami Harju
Ajax-sovellusten suunnittelu ja toteutus
Marraskuu 2008

40 sivua

Tampereen ammattikorkeakoulu
Viestinnän koulutusohjelma
Vuorovaikutteisuuden suunnittelu
Lopputyön muoto: Kirjallinen
Lopputyön ohjaaja: Ari Närhi
Avainsanat: Ajax-ohjelmointi, Internet

Opinnäytetyö käsittelee Ajaxia, web-tekniikkaa tai lähestymistapaa web-suunnitteluun, joka yhdistää useita vanhoja, jo pitkään käytössä olleita teknologioita yhteen ja mahdollistaa asynkronisen tietoliikenteen selaimen ja palvelimen välillä. Se lähestyy aihetta historiallisesta, suunnittelullisesta sekä Ajaxin kehitystyöhön liittyvien haasteiden ja ratkaisumallien näkökulmasta. Työn tekninen osuus tarkastelee Ajaxin keskeisimmän komponentin, XMLHttpRequest-objektin ja sen käyttämien dataformaattien toimintaa. Ajaxin avulla voidaan kehittää dynaamisia web-käyttöliittymiä ja -sovelluksia, jotka ovat käyttökokemukseltaan työpöytäsovellusten kaltaisia. Työn tulokset osoittavat Ajaxin olevan tällä hetkellä käyttökelpoisin, selainlisäosia vaatimaton tekniikka tämänkaltaisia internet-sovellusten kehittämiseen.

THESIS SUMMARY

Sami Harju
The design and realization of Ajax applications
November 2008

40 pages

TAMK University of Applied Sciences
Media Programme
Area of specialisation: Interaction Design
Type of Final Project: Written
Thesis supervisor: Ari Närhi
Keywords: Ajax programming, Internet

The thesis is about Ajax, a web technique or a comprehensive approach to web design, in which a variety of old technologies that have already been used for a long time converge and which makes asynchronous communication between the browser and the server possible. It deals with the subject from a historic, design and development challenge and solution point of view. The technically oriented part of the thesis studies the functionality of Ajax’s most essential component, the XMLHttpRequest object, and the data formats it utilizes. Ajax allows a developer to create more dynamic web interfaces and applications that are closing the gap between internet and desktop applications in terms of usability and user experience. The research results show that Ajax is currently the most feasible plug-in free technique for developing this kind of internet applications.

Sisällys

1 Johdanto........5
2 Taustaa Ajaxista........7
2.1 Historiallinen viitekehys........7
2.2 Mitä Ajax on?........8
2.3 Mitä Ajax ei ole?........11
3 XMLHttpRequest........13
3.1 Instanssin luominen........13
3.2 Pyynnön lähettäminen........15
3.2.1 onreadystagechange........16
3.2.2 open........16
3.2.3 send........17
3.2.4 setRequestHeader........17
3.2.5 Esimerkki Ajax-pyynnöstä........17
3.3 Vastaukseen reagointi........18
3.3.1 readyState........19
3.3.2 status........19
3.3.3 responseText & responseXML........20
3.3.4 Esimerkki palvelimen vastausta tarkastelevasta funktiosta........20
4 Dataformaatit tiedonsiirrossa........21
4.1 Tekstimuotoiset vastaukset........21
4.2 XML........21
4.2.1 Esimerkki XML-muotoisen vastauksen käsittelystä........22
4.3 JSON........23
4.3.1 eval........24
4.3.2 JSON-parserit........25
4.4 HTML........25
5 Suunnittelu ja implementointi........27
5.1 Ajax ja saavutettavuus........28
5.1.1 Progressive enhancement........29
5.1.1.1 Unobtrusive JavaScript........30

6 JavaScript-kirjastot..32

7 Case-esimerkki: Audionce...34

7.1 Perustoiminnallisuuden suunnittelu ja toteutus.....................................34

7.2 Toiminnallisuuskerroksen implementointi...36

7.3 Johtopäätökset...37

8 Yhteenveto...39

Lähteet...41

1 Johdanto

Ajax on vuosituhannen alussa muotoutunut ja vuonna 2005 nimetty web-tekniikka, joka kokoaa useita vanhoja, jo pitkään käytössä olleita teknologioita yhteen mahdollistaen uudenlaisia ratkaisumalleja web-kehityksessä. Sen keskeisin ominaisuus on kyvykkyys asynkroniseen tietoliikenteeseen selaimen ja palvelimen välillä, joka rikkoo internetin perinteisen koko sivun latauksiin perustuvan toiminnan kulun. Tämä tarkoittaa käytännössä sitä, että web-käyttöliittymistä on mahdollista rakentaa entistä paremmin reagoivia, luonteeltaan rikkaampia ja ominaisuuksiltaan sekä käyttötuntumaltaan työpöytäsovellusten kaltaisia. Ajax on siis pienentänyt merkittävästi eroa työpöytä- ja internet-sovellusten välillä – yksi palvelukeskeisen Web 2.0 -konseptin, jonka lippulaivaksi Ajax on usein nimetty, olennaisimpia teemoja onkin ajatus internetistä julkaisualustana. (O'Reilly 2005)

Ajax on noussut muutamassa vuodessa yhdeksi merkittävimmistä ja samalla muodikkaimmista ilmiöistä web-kehityksen piirissä. Sen erilaisten sovelluskohteiden kirjo on valtavan laaja, ja itse asiassa nykyisin valtaosa tuoreista palveluista internetissä hyödyntää Ajaxia tavalla tai toisella. Yksi sen ominaispiirteitä onkin kyky taipua hienovaraisista käyttöliittymäparannuksista aina täysveristen sovellusten ydinosaksi. Ajaxin merkityksellisyydestä kertoo varmasti jotakin myös se, että sen synnystä asti sitä eteenpäin vievän rintaman eturivissä on marssinut muun muassa Google suositun, web-maailmassa mullistuksia tehneen tuoteperheensä kanssa.

Työssäni pyrin antamaan aiheesta kiinnostuneelle lukijalle yleiskatsauksen Ajaxista – mitä se on, mitä sillä voidaan tehdä ja kuinka se toimii. Tarkoituksenani ei ole kartoittaa koko Ajaxin mahdollisuuksien kattamaa kenttää, mikä tuskin olisi käytännöllisesti katsoen edes mahdollista ottaen huomioon uudenlaisten sovellusten nykyisen kehitystahdin. Sen sijaan teksti tarjoaa eväät aiheen jatko-opiskelulle perehdyttämällä lukijan Ajaxiin liittyviin peruskysymyksiin. Käyn läpi myös joitakin Ajax-sovellusten suunnitteluun ja toteutukseen liittyviä haasteita ja pyrin antamaan niihin vastauksia.

Lähestyn aihetta yhtäältä puhtaasti teknisestä, toisaalta yleisellä tasolla suunnittelullisesta sekä kolmanneksi Ajax-kehitykseen liittyvien haasteiden ja ongelmien sekä niiden ratkaisumallien näkökulmista. Lopuksi kokoan yhteen omia

kokemuksiani ja havaintojani aiheesta käytännön case-esimerkin kautta. Aivan ensimmäisenä on kuitenkin tarpeellista luoda katsaus Ajaxin historialliseen viitekehykseen – mistä se on syntynyt?

2 Taustaa Ajaxista

2.1 Historiallinen viitekehys

Internetin alkujuurilta, tiedemiesten ja tutkijoiden dokumenttien sekä tutkimustulosten vaihtopaikasta, on kuljettu pitkä matka tämän päivän käyttäjä- ja palvelukeskeiseen globaaliin tietoverkkoon, joka saavuttaa lähes 1,5 miljardia ihmistä (Internet World Stats) ympäri maailman. Internetin luonteen muutosta on tietysti vauhdittanut tarvittavan tekniikan kehitys ja yleistyminen, mutta vielä tähdellisempää lienee webin valjastaminen markkinakäyttöön, uudeksi kanavaksi kaikenlaiselle kaupankäynnille sekä mainostamiselle, sekä sen suosion ja käyttäjämäärän kasvu.

Alun alkaen kaikki sivustot olivat luonteeltaan staattisia – käyttäjä pyysi jotakin resurssia ja palvelin lähetti sen käyttäjälle. Tämä malli ei mahdollistanut kovinkaan monimutkaisten sovellusten kehittämistä. Hiljalleen webiin alkoi kehittyä yksinkertaisiin lomakkeisiin pohjautuvaa toiminnallisuutta, joka yhä edelleen toimii perustana suurimmalle osalle web-sovelluksien toiminnallisuudesta. Tämä mahdollisti muun muassa erilaisten yhteydenottolomakkeiden sekä verkkokauppojen ja keskustelufoorumeiden kehittämisen.

Webin sisällöt alkoivat siis muuttua luonteeltaan dynaamisemmiksi ja enemmän käyttäjäkeskeisiksi erityisesti vuosituhannen vaihteen molemmin puolin. Erilaiset teknologiat, kuten Sun Microsystemsin kehittämällä Java-ohjelmointikielellä luodut web-sovelmat ja myöhemmin Macromedian (nykyisin Adoben) kehittämä Flash, alkoivat yleistyä ja vastasivat 1990-luvun puolivälistä lähtien käyttäjien vaatimuksiin entistä toiminnallisempia sivustoja ja sovelluksia kohtaan. Yhteinen ongelma näillä teknologioilla on niiden tarve ylimääräiselle selainlisäosalle tai sovellusalustalle, mikä luonnollisesti vaikuttaa negatiivisesti sovellusten tavoitettavuuteen, ja joka erityisesti Javan kohdalla on aiheuttanut teknisiä ongelmia alustan versioiden erojen vuoksi.

Käyttäjät siis vaativat entistä toiminnallisempia sovelluksia verkkoon ja kehittäjät halusivat päästä eroon jokaisen potentiaalisen käyttäjän koneelle jaettavista asennuspaketeista ja täten saavuttaa mahdollisimman laajan kohderyhmän sivustoilleen sekä sovelluksilleen. Vastaus ongelmaan löytyisi selaimien valmiiksi ymmärtämiin

standardisoituihin teknologioihin perustuvasta tekniikasta. (Asleson & Schutta 2006, 14 & 22)

2.2 Mitä Ajax on?

Vuonna 2005 Adaptive Pathin perustaja Jesse James Garrett kirjoitti artikkelissaan *Ajax: A New Approach to Web Applications* vuorovaikutteisuuden suunnittelun nykytilasta, ja siitä kuinka valtaosa innovatiivisesta interaktion suunnittelusta tapahtuu nykyisin web-sovellusten ympärillä. Web-sovellukset olivat alkaneet muuttua kohti entistä rikkaampia käyttöliittymiä ja joustavampaa käytettävyyttä pienentäen kuilua perinteisten työpöytäsovellusten ja internetiä alustanaan käyttävien sovellusten välillä. Muutoksen mahdollistavalle, uudelle lähestymistavalle hän antoi artikkelissaan nimen Ajax. Aluksi se oli yksinkertaisesti lyhenne sanoista Asynchronous XML + Javascript (AJAX), mutta myöhemmin termin merkitys on laventunut kattamaan alleen useita erilaisia tekniikoita ja niiden erilaisia yhdistelmiä – kuvaamaan enemmänkin kokonaisvaltaista lähestymistapaa sovellus- ja yleisen tason suunnitteluun internetissä, ja edustamaan perinpohjaista muutosta sivustojen sekä sovellusten mahdollisuuksissa.

Garrett tähdensi kyseessä olevan itse asiassa yhdistelmä vanhoja, jo käytössä olevia teknologioita kokonaan uuden teknologian sijaan. Hänen määritelmänsä mukaan Ajax yhdistää:

- standardien mukaisen esitystavan hyödyntäen XHTML:ää sekä CSS:ää
- Document Object Modelin (tästä eteenpäin DOM(-malli)) datan dynaamiseen esittämiseen ja interaktion sekä toiminnallisuuden toteuttamiseen
- XML:n ja XSLT:n datan molemmin puoliseen (palvelin-asiakas) siirtoon ja muokkaamiseen
- asynkronisten hakujen tekemisen palvelimelle XMLHttpRequest-objektin (tästä eteenpäin XHR-objekti) avulla
- JavaScriptin, joka lopulta sitoo kaiken yhteen

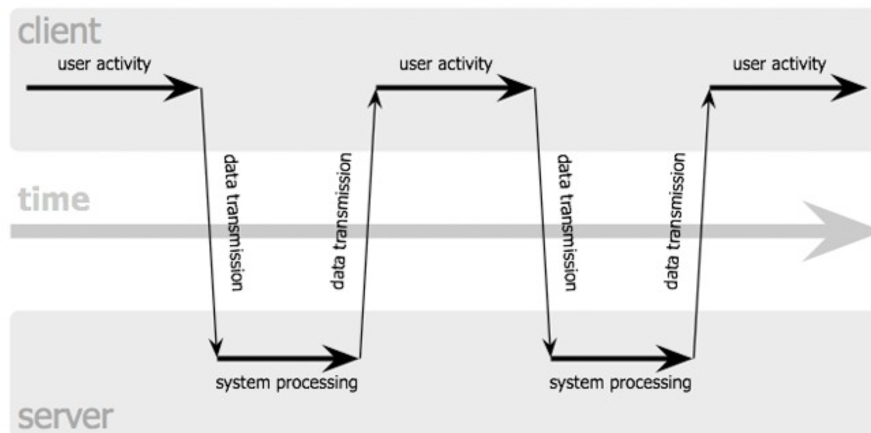
Myöhemmin Garrett kuitenkin tarkensi, ettei Ajaxin käyttö edellytä kaikkien näiden teknologioiden hyödyntämistä, vaan esimerkiksi tiedonsiirrossa ja -käsittelyssä datamuotona voidaan käyttää erilaisia vaihtoehtoja XML:lle. XHR-objektia sekä tiedonsiirtoformaatteja käsittelen myöhemmin syvällisemmin omissa luvuissaan.

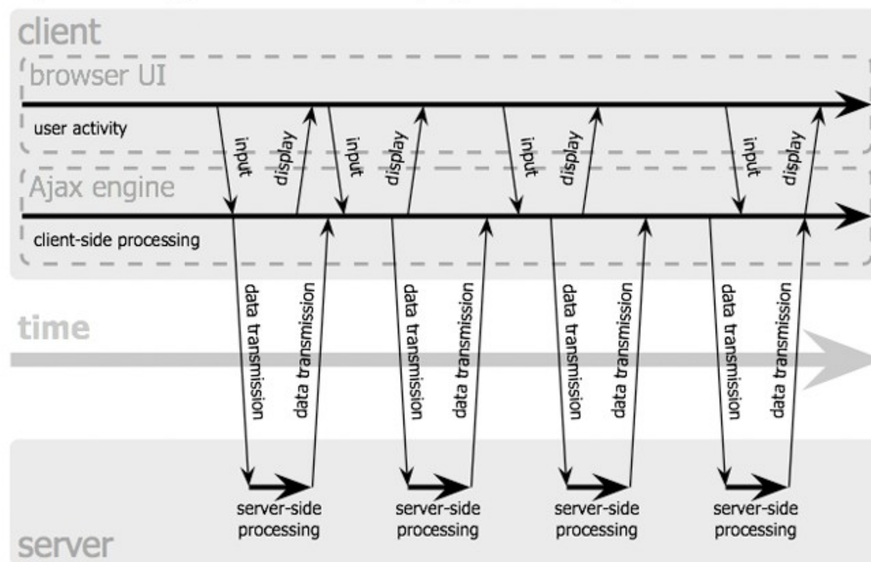
Vaikka sana *Ajax* ei toimikaan lyhenteenä, on se silti erinomainen tapa yhdistää koko joukko yhdessä toimivia teknologioita yhteen sanaan. Sen tarpeellisuudesta voidaan kiistellä, mutta joka tapauksessa on selvää, että se on avuksi niin kehittäjien kuin asiakkaidenkin välisessä keskustelussa koskien tärkeitä käytettävyyteen ja designiin liittyviä näkökantoja modernissa web-suunnittelussa. (Keith 2007, 4)

Ajax ei siis sinällään ole mitään uutta, vaan oikeastaan vain uusi nimi kokoelmalle vanhoja teknologioita, jotka yhdistyvät tehokkaalla, uudella tavalla. Sen tuoreinta teknologiaa edustaa XHR-objekti, joka julkaistiin ActiveX-komponenttina alun perin Internet Explorerin viidennen version myötä keväällä 1999. Sen tehokas käyttöönotto on kuitenkin mahdollistunut vasta 2000-luvun alkuvuosien jälkeen merkittävän markkinaosuuden omistavien Mozilla- ja Safari-selainten alettua tukea sen käyttöä. Ajaxin suosion kasvamista on osaltaan merkittävästi edesauttanut myös selainten kehitys pois selainspesifisistä ominaisuuksista, joita räätälöitiin erityisesti 1990-luvun lopulla Netscape Navigatorin ja Internet Explorerin kilpaillessa valta-asemasta selainmarkkinoilla, kohti yhteisiä, World Wide Web Consortiumin (W3C) asettamia web-teknologioita koskevia standardeja 2000-luvun alkuvuosien aikana. Aiemmin muun muassa JavaScript on ollut varsin huonossa valossa kehittäjien keskuudessa vaihtelevan selaintuen vuoksi, ja CSS:n, HTML:n sekä DOM-mallin kohdalla eri selainten poikkeavat tulkintatavat ovat vaikeuttaneet Ajax-sovellusten kehittämistä. Nykytilanne alkaa kuitenkin näyttää jo hyvältä, ja sikäli kun nykyinen trendi selainkehityksessä jatkuu samanlaisena, näyttää tulevaisuus entistä valoisammalta. Tänä päivänä eri selaimia on kuitenkin otettava huomioon tavalla tai toisella Ajaxia käytettäessä, kuten muun muassa XHR-objektia käsittelevässä luvussa tulemme huomaamaan. Yhdeksi hyvän Ajax-sovelluksen tuntomerkiksi voidaan kuitenkin nimetä selainriippumattomuus, jonka saavuttamisen mahdollistaa standardeja noudattavan koodin kirjoittaminen.

Huolimatta siitä, että Ajaxin yhdistämät teknologiat ovat itsessään verrattain vanhoja, tarjoaa Ajax merkittävästi perinteisestä mallista poikkeavan lähestymistavan internetissä totuttuun pyyntöjen ja vastausten varaan rakentuvaan liiketoimintalogiikkaan, jossa käyttäjä suorittaa sivulla haluamansa toiminnon ja lähettää pyynnön palvelimelle, joka puolestaan palauttaa takaisin kokonaan uudestaan ladatun sivun pyynnön mukaisesti muuttuneella sisällöllä. Ajax mahdollistaa asynkronisten pyyntöjen lähettämisen ja vastausten saamisen palvelimelta, joka puolestaan tuo web-sovelluksen käyttökokemuksen lähemmäs työpöytäsovelluksista tuttua jouhevuutta ja sovellusten dynaamista luonnetta. Käytännössä asynkronisen kutsun avulla käyttäjä voi siis pyytää vain tarvittavaa osaa sivusta päivitettäväksi ja jatkaa työskentelyään palvelimen suorittaessa pyyntöä ja ladatessa uutta sisältöä käyttäjän selaimeen, toisin kuin perinteisessä internet-maailman klikkaa-odota-klikkaa-odota -skenaariossa. (Asleson & Schutta 2006, 14–15) Asynkronisten kutsujen tekemiseen palvelimelle on aiemminkin ollut tekniikoita, joista on käytetty muun muassa *remote scripting* nimitystä, mutta ne ovat usein olleet luonteeltaan hack:ien kaltaisia ja enemmän tai vähemmän väliaikaisia ratkaisuja odotellessa parempaa selaintukea XHR-objektille. En käsittele niitä tekstissäni tämän enempää.

Se, että Ajax pohjautuu jo olemassa oleviin teknologioihin, on itse asiassa yksi sen suurista vahvuuksista. Näitä teknologioita on kehitetty ja hyödynnetty useiden vuosien ajan, ja ne ovat siten jo entuudestaan tuttuja monille kehittäjille ja niille on useimmiten olemassa hyvä dokumentaatio sekä aktiivinen käyttäjäyhteisö, jonka panos näkyy internetissä lukemattomina sovellusten kehittämistä helpottavina koodiesimerkkeinä, tutoriaaleina ja jopa koodikirjastoina. Täten Ajaxin oppimiskynnys on matala ja käyttöönotto helppoa. Ajaxia, tai tarkemmin sen hyödyntämää XHR-objektia, voidaan luonnehtia sillaksi selaimen ja palvelimen välillä, ja se voidaan yhdistää mihin tahansa kehittäjän vapaasti valitsemaan palvelinpuolen teknologiaan. Myös tämä osaltaan helpottaa Ajaxin käytön aloittamista tai esimerkiksi implementointia jo olemassa oleviin palvelinpuolen ratkaisuihin. Siinä, missä Ajaxin kilpailijat, kuten Java-sovelmat sekä Flash, tarvitsevat oman selainlisäosan tai ympäristön toimiakseen, toimii potentiaalisesti Ajax käytännössä lähestulkoon kaikissa moderneissa selaimessa ilman minkäänlaisia lisäosia tai asennusprosessia.





Garrettin (2005) kaavio synkronisen ja asynkronisen web-sovelluksen toiminnasta.

Keith kiteyttää kirjansa, *Bulletproof Ajax*, avausluvussa (s.6) Ajaxin määritelmän selkokielisesti ja osuvasti: se on ”tapa kommunikoida palvelimen kanssa päivittämättä koko sivua.”

2.3 Mitä Ajax ei ole?

Kun termi *Web 2.0* esiteltiin vuonna 2004, se nousi nopeasti kehittäjien sekä markkinaihmisten huulille ja sen käyttö vakiintui nopeasti. Sen sisällöstä ja merkityksestä käytiin kuitenkin paljon kiistelyä, eikä ollut vaikeaa osoittaa, kuinka

termiä käytettiin joko turhan kevein tai täysin tuulesta temmatuin perustein. Vain vajaa vuosi myöhemmin termi *Ajax* alkoi läpikäydä hyvin samankaltaista prosessia. Sitä on sittemmin käytetty väärin tietoisesti markkinointikikkana, kuten useita muita uusia alan trendikkäitä termejä. Merkittävin osa Ajaxin ympärillä pyörivästä kohusta johtuu suurista, tekniikkaa voimakkaasti hyödyntävistä sovelluskehittäjistä, kuten Googlesta, ja heidän jättimäisen suosion saavuttamista Ajax-pohjaisista palveluistaan, kuten Gmail tai Google Maps. (Asleson & Schutta 2006, 14)

Yleisimmin Ajaxiin näkee virheellisesti viitattavan puhuttaessa yksinkertaisesti DOM-skriptauksesta tai DHTML:stä (Dynamic HTML), jotka käytännössä tarkoittavat vain staattiseen HTML-rakenteeseen toiminnallisuuden lisäämistä tyypillisimmin JavaScriptin avulla. DOM-skriptaamalla saavutetaan HTML-sivulla helposti Ajaxia muistuttavaa toiminnallisuutta, mutta todellisuudessa sivulle päivitettävät sisällöt tai muu käyttäjän toiminnan aiheuttama muutos tuodaan dynaamisesti esiin jo selaimeen sivun avaamisen yhteydessä ladatusta sisällöstä. Vaikka DOM-skriptaaminen on keskeinen osa Ajax-periaatteen mukaista datan käsittelyä, eikä Ajaxia olisi olemassa ilman mahdollisuutta muokata dokumentin tietorakennetta dynaamisesti, ei se kuitenkaan ole kuin yksi osa tässä kokonaisuudessa.

On totta, että lähes jokainen web-sovellus voi hyötyä Ajaxin käytöstä, mutta tämä ei kuitenkaan tarkoita Ajaxin olevan oikea lähestymistapa kaikkiin ratkaisuihin. Pahimmillaan huonosti toteutettu Ajax-sovellus saattaa huonontaa sivuston saavutettavuutta ja estää käyttäjän pääsyn käsiksi haluamaansa sisältöön, tai muuten vain tehdä tavallisesti yksinkertaisen toiminnon suunnittelusta ja toteutuksesta turhan monimutkaista. Ajax ei ole ratkaisu kaikkeen. (Asleson & Schutta 2006, 22; Holdener III 2008, 34)

3 XMLHttpRequest

XMLHttpRequest (XHR) -objekti mahdollistaa asynkronisen kommunikoinnin selaimen ja palvelimen välillä. Se on Ajaxin sydän. Ajaxin suosion kasvaessa räjähdysmäisesti vuodesta 2005 alkaen, se liitettiin myös osaksi W3C:n spesifikaatiota jo vuonna 2006. Tämä tapahtui epätavallisen nopeasti ottaen huomioon, että XHR-objekti kehitettiin alun perin Microsoftin toimesta osaksi Internet Explorerin 5-versiota vuonna 1999 ja otettiin käyttöön muissa valtavirtaisissa selaimissa vasta 2000-luvun puoliväliin mennessä. Internet Explorerissa XHR-objekti toteutettiin ActiveX-komponenttina selainversioissa 5 ja 6, kun taas muut valmistajat implementoivat sen natiiviksi osaksi selaintaan. Microsoft päätti luopua ActiveX-lähestymistavasta, ja XHR-objekti on natiivi osa myös Internet Exploreria versiosta 7 lähtien.

Selainversiokohtaiset erot on edelleen syytä ottaa huomioon XHR-objektia alustettaessa, jotta voidaan varmistaa sitä hyödyntävälle sovellukselle mahdollisimman laaja selaintuki. Täytyy kuitenkin muistaa, että vaikka käytössä olisi teknisesti Ajaxiin kykenevä selain, voi käyttäjä kytkeä JavaScriptin pois päältä, jolloin XHR-objektin alustaminen ei ole mahdollista, eikä Ajax siis toimi. Internet Explorerin tapauksessa (versiot 5 – 6) myös ActiveX-tuki voi olla kytkettynä pois päältä, joka niin ikään estää Ajaxin toiminnan. (Keith 2007, 47) XHR-objektia vaativaa toiminnallisuutta suunniteltaessa on siis pidettävä mielessä se, että Ajax ei ole jokaiselle käyttäjälle vaihtoehto. Mikäli halutaan saavuttaa mahdollisimman laaja kohderyhmä, on Ajax-toiminnallisuuksille tehtävä JavaScriptitön vaihtoehto. Käsittelen tätä problematiikkaa tarkemmin viidennessä luvussa.

Tässä luvussa käyn läpi XHR-objektin käytön pääpiirteissään suurilta osin seuraten Jeremy Keithin kirjassaan Bulletproof Ajax käyttämää lähestymistapaa sen selkokielisyyden ja hyvien esimerkkien vuoksi.

3.1 Instanssin luominen

XHR-objektin alustaminen useimmissa selaimissa on melko suoraviivainen prosessi. Tarvitsee vain luoda JavaScript-muuttuja, johon sijoitetaan uusi ilmentymä XMLHttpRequest-objektista:

```
var xhr = XMLHttpRequest();
```

Internet Explorerissa tämä on kuitenkin hieman monimutkaisempaa ActiveX-lähestymistavan vuoksi:

```
var xhr = ActiveXObject(”Msxml2.XMLHTTP”);
```

Lisäksi jotkut Internet Explorerin versioista käyttävät vanhempaa versiota kyseisestä ActiveX-komponentista:

```
var xhr = ActiveXObject(”Microsoft.XMLHTTP”);
```

Jotta saisimme XHR-objektin alustamisen toimimaan suurimmalla todennäköisyydellä kaikissa teknisesti Ajaxiin kykenevissä selaimissa, on hyvä luoda oma funktio objektin alustamista varten. Keithin käyttämä esimerkki muokkaa W3Schools-sivustolla, osoitteessa http://www.w3schools.com/Ajax/ajax_browsers.asp, esitettyä funktiota hieman asettamalla sen palauttamaan alustetun XHR-objektin suoraan käyttäjän luomaan muuttujaan:

```
function getHTTPObject() {
  var xhr = false;
  if(window.XMLHttpRequest) {
    xhr = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    try {
      xhr = new ActiveXObject(”Msxml2.XMLHTTP”);
    } catch(e) {
      try {
        xhr = new ActiveXObject(”Microsoft.XMLHTTP”);
      } catch(e) {
        return false;
      }
    }
  }
  return xhr;
}
```

Tämän funktion avulla XHR-objektin selainriippumaton alustus tapahtuu siis seuraavasti:

```
var myHXR = getHTTPObject();
```

Funktiossa käytetään objektin tunnistusta (engl. object detection) ActiveX:n olemassa olon havaitsemiseen. Sen mukaan päätetään, millä tavalla XHR-objekti alustetaan. En mene tarkemmin käytetyn JavaScript-syntaksin selittämiseen, mutta muun muassa *try ... catch* -rakenteen toimintaan voi tutustua esimerkiksi WCSchoolsin sivustolla osoitteessa http://www.w3schools.com/js/default.asp.

Selainkohtaiset eroavaisuudet XHR-objektin käytössä loppuvat objektin alustamisen jälkeen. Tästä eteenpäin sen metodeihin ja ominaisuuksiin päästään käsiksi samalla tavalla riippumatta siitä, onko se selaimen natiivi objekti vai ActiveX-objekti.

3.2 Pyynnön lähettäminen

Palvelimelle pyyntöä XHR-objektin kautta lähetettäessä käytetään usein seuraavia metodeja:

Metodi	**Kuvaus**
open(*pyynnön tyyppi, url, asynk, nimi, salasana*)	- *Pyynnön tyyppi* määrittelee pyynnössä käytettävän metodin: se voi olla esimerkiksi *GET* tai *POST* - *Url* on palvelimelta pyydettävän tiedoston osoite absoluuttisessa tai suhteellisessa muodossa - *Asynk* on boolean-arvo, joka määrittää lähetetäänkö pyyntö asynkronisena vai synkronisena, oletusarvo *true* - Harvemmin käytettävät parametrit *nimi* ja *salasana* määrittävät palvelimelle pyynnön yhteydessä lähetettävät käyttäjätiedot
send(*sisältö*)	- *Sisältö* on palvelimelle pyynnön yhteydessä lähetettävä merkkijono
setRequestHeader(*otsake, arvo*)	- *Otsake* määrittää lähettävän datan mukana lähetettävän otsaketiedon nimen - *Arvo* määrittää lähetettävän otsaketiedon arvon

(W3Schools, XMLHttpRequest; Asleson & Schutta 2006, 27)

XHR-objektin alustamisen jälkeen voidaan valmistella ja lähettää pyyntö palvelimelle. Prosessissa käytetään pääosin neljää yllä esiteltyä XHR-objektin osaa.

3.2.1 onreadystagechange

onreadystatechange on JavaScript-tapahtumankäsittelijä, joka voidaan laittaa osoittamaan mihin tahansa käyttäjän määrittelemään JavaScript-funktioon. Toisin kuin monet tutut käyttäjän laukaisemia tapahtumia kaappaavat tapahtumankäsittelijät, kuten lomakkeen lähettämisen yhteydessä käytettävä *onsubmit* tai muun muassa paljon käytetyt hiiren liikkeen kaappaamiseen käytetyt *onmouseover* ja *onmouseout*, käsittelee *onreadystatechange* palvelimen laukaisemia, lähetetyn pyynnön edetessä muuttuvia *readyState*-tapahtumia. Sen avulla käynnistetään tavanomaisesti funktio tai funktioita, jotka lähetetyn pyynnön valmistuttua parsivat vastaanotetun datan ja päivittävät sen sivulle. (Keith 2007, 51; Asleson & Schutta 2006, 29)

3.2.2 open

open-metodilla alustetaan varsinainen palvelimelle lähetettävä pyyntö. Se määrittää palvelimelta haettavan sivun sijainnin sekä käytettävän metodin, samaan tapaan kuin normaalisti sivuja avattaessa web-selain tekee. Vaikka metodin valintaan ei ehdotonta oikeata vastausta ole, niin yleisesti *GET*-metodia suositellaan käytettävän, kun palvelimelta voin noudetaan dataa. *POST*:ia sen sijaan käytetään dataa palvelimelle lähetettäessä. *GET* käy hyvin myös usein pienen datamäärän siirtoon, mutta sillä ei tulisi ikinä lähettää tietoa palvelimelle, jos se muuttaa palvelimen tilaa esimerkiksi lisäten tai poistaen jotakin sinne talletettua tietoa.

Metodin parametrina voidaan myös syöttää tieto siitä, tehdäänkö pyyntö asynkronisena vai ei. Ajax-pyynnöt ovat oletusarvoisesti asynkronisia, sillä synkronisessa pyynnössä skriptin suorittaminen loppuisi siksi aikaa, kun vastaus palvelimelta on saatu ja täten menetettäisiin useimmiten Ajaxilla tavoiteltavan jouhean käyttökokemuksen tuntu. Parametreina on mahdollista syöttää myös kutsun yhteydessä käytettävä käyttäjätunnus sekä salasana, mutta niitä ei luonnollisesti kannata kirjoittaa JavaScript-tiedostoon, joka on kaikkien nähtävillä. Sen sijaan käyttäjän tunnistus hoidetaan tavallisesti kokonaan suljetusti palvelimen päässä. (Keith 2007, 52; Asleson & Schutta 2006, 27; W3Schools, XMLHttpRequest)

3.2.3 send

send-metodilla lähetetään valmisteltu pyyntö palvelimelle. Se ottaa parametrikseen palvelimelle lähetettävän datan merkkijonona. *GET*-metodia käyttäessä, eli kun palvelimelle ei haluta lähettää mitään dataa, vaan ainoastaan noutaa aiemmin määritelty tiedosto, parametriksi annetaan *null*. Sen sijaan dataa palvelimelle lähetettäessä, metodin ollessa *POST*, data muotoillaan URL-enkoodatuksi kyselymerkkijonoksi, jossa nimi-arvo -parit yhdistetään &-merkein. (Keith 2007, 55) Esimerkiksi näin:

```
nimi=Sinun+nimesi&viesti=Sinun+viestisi
```

3.2.4 setRequestHeader

setRequestHeader-metodia ei tarvitse käyttää *GET*-metodia käytettäessä. Sen sijaan *POST*:in yhteydessä palvelimelle lähetettävälle pyynnölle on määriteltävä sen mukana kulkevan datan tyyppi. Yllä olevan kyselymerkkijonon yhteydessä headerin otsakkeeksi annettaisiin *Content-type* ja arvoksi *application/x-www-form-urlencoded*. (Keith 2007, 55; Asleson & Schutta 2006, 28) Erilaisiin otsaketyyppeihin en paneudu tämän enempää.

3.2.5 Esimerkki Ajax-pyynnöstä

Pyynnön lähettäminen palvelimelle aiemmin luotua *getHTTPObject*-funktiota XHR-objektin alustamiseen hyödyntäen voisi tapahtua esimerkiksi näin:

```
var myRequest = getHTTPObject();
  if(myRequest) {
    myRequest.onreadystatechange = parseResponse;
    myRequest.open(”POST”, ”formHandler.php”);
    myRequest.setRequestHeader(”Content-type”, ”application/x-www-
form-urlencoded”);
    myRequest.send(”nimi=Sami+Harju&viesti=Hello+world”);
}
```

- Luodaan uusi instanssi XHR-objektista:
  `var myRequest = getHTTPObject();`

- Tarkistetaan, että objekti on käytettävissä:
  `if(myRequest) {`

- Asetetaan *parseResponse* -niminen funktio *onreadystatechange*-tapahtumankuuntelijaan:
```
myRequest.onreadystatechange = parseResponse;
```

- Alustetaan yhteys käyttäen *POST*-metodia, haettavaksi tiedostoksi asetetaan *formHandler.php*:
```
myRequest.open(”POST”, ”formHandler.php”);
```

- Asetetaan header-tieto siitä, että palvelimelle ollaan lähettämässä URL-enkoodattua lomaketietoa:
```
myRequest.setRequestHeader(”Content-type”, ”application/x-www-form-urlencoded”);
```

- Lopuksi lähetetään valmisteltu kutsu. Kutsun mukana URL-enkoodattu merkkijono:
```
myRequest.send(”nimi=Sami+Harju&viesti=Hello+world”);
}
```

3.3 Vastaukseen reagointi

Palvelimen lähettämän vastauksen kuuntelemiseen käytetään seuraavia palvelimen päivittämiä XHR-objektin ominaisuuksia:

Ominaisuus	**Kuvaus**
readyState	- Ilmoittaa Ajax-pyynnön sen hetkisen tilan. Sen arvo on numeerinen: * 0 Uninitialized. *open*-metodia ei vielä ole kutsuttu * 1 Loading. *open*-metodia on kutsuttu, *send*-metodia ei. * 2 Loaded. *send*-metodia on kutsuttu, pyyntö on lähetetty. * 3 Interactive. Palvelin on lähettämässä vastausta. * 4 Complete. Vastaus on lähetetty.
status	- Palvelimen lähettämä kolmi-numeroinen HTTP status-koodi, joka ilmoittaa palvelimen tai pyydetyn tiedoston

	tilan.
responseText	- Palvelimen lähettämä vastaus merkkijonona.
responseXML	- Palvelimen lähettämä XML-muotoinen vastaus.

(W3Schools, XMLHttpRequest; Asleson & Schutta 2006, 29)

3.3.1 readyState

readyState on ominaisuus, joka päivittyy palvelimen toimesta Ajax-pyynnön etenemisen eri vaiheissa. Se laukaisee XHR-objektin *onreadystatechange*-tapahtumankäsittelijän aina muuttuessaan, eli jos tähän tapahtumankäsittelijään on osoitettu funktio, se suoritetaan joka kerta. Teoreettisesti *readyState* muuttaa numeerista arvoaan järjestyksessä nollasta neljään, mutta käytännössä tämä vaihtelee selainkohtaisesti. Selaimesta riippumatta pyynnön päätyttyä *readyState*:n arvo kuitenkin on 4. On siis turvallisinta odottaa palvelimen palauttaessa *readyState*:n arvolla 4, ennen kuin mitään toimintoja selaimessa aletaan suorittaa. (Keith 2007, 56)

3.3.2 status

Samaan tapaan kuin lähetettävien pyyntöjen yhteydessä, myös palvelimen lähettämien vastausten mukana tulee erilaista header-tietoa sekä palvelimesta että palautettavasta datasta – muun muassa palautettavan dokumentin tyypin (HTML, XML yms.) sekä merkistö-enkoodauksen (*utf-8* yms.) muodossa. Tärkein header-tieto on kuitenkin palvelimen lähettämä kolmi-numeroinen status-koodi. Se on osa HHTP-protokollaa. Kaikkein tutuin status-koodi lienee *404* (Not Found), joka tarkoittaa, että pyydettyä dokumenttia ei löytynyt. Yleisin koodi kuitenkin on *200* (OK), joka ilmoittaa onnistuneesta vastauksesta. Status-koodi ehdollisen *GET*-pyynnön yhteydessä on *304* (Not Modified), joka kertoo pyydetyn dokumentin tilan pysyneen muuttumattomana edellisen pyynnön jälkeen – myös *304* statusta varten on syytä laittaa tarkistus onnistuneen vastauksen käsittelyn yhteyteen. (Keith 2007, 57; W3Schools, XMLHttpRequest)

3.3.3 responseText & responseXML

responseText-ominaisuus sisältää XHR-objektille palautetun datan merkkijonona. Riippuen siitä, mitä palvelin on lähettänyt, tämä voi sisältää esimerkiksi merkkijonon HTML:ää tai mitä vain tekstuaalista dataa. Mikäli palvelin palauttaa pyydetyn datan XML-oliona, se on luettavissa *responseXML*-ominaisuuden kautta. *responseText*- sekä *responseXML*-ominaisuudet ovat kokonaisuudessaan selaimen luettavissa, kun palvelin on palauttanut *readyState*-ominaisuuden arvolla 4 – sitä ennen niiden käsittelyä ei tulisi aloittaa. (Keith 2007, 59) Käsittelen molempia ominaisuuksia tarkemmin tiedonsiirtoformaatteja käsittelevässä luvussa.

3.3.4 Esimerkki palvelimen vastausta tarkastelevasta funktiosta

Aiemmassa esimerkissä asetin *onreadystatechange*-tapahtumankäsittelijään viittamaan *parseResponse* nimiseen omaan funktioon. Kyseinen funktio voisi näyttää tällaiselta:

```
function parseResponse() {
  if(myRequest.readyState == 4) {
    if(myRequest.status == 200 || myRequest.status == 304) {
      alert(myRequest.responseText);
    }
  }
}
```

- Koska kyseinen funktio suoritetaan joka kerta, kun palvelin lähettää tiedon muuttuneesta *readyState*:sta, on ensimmäiseksi tarkistettava, että vastaus on lähetetty, eli koodi on 4:
  ```
  if(myRequest.readyState == 4) {
  ```

- Tarkastetaan, että palvelimen lähettämä status-koodi on *200* tai *304*, eli vastaus on onnistunut:
  ```
  if(myRequest.status == 200 || myRequest.status == 304) {
  ```

- Tämän jälkeen palvelimen lähettämälle vastaukselle voidaan tehdä mitä halutaan lukemalla se *responseText*-, tai jos se on XML-muodossa, *responseXML*-ominaisuuden kautta. Esimerkissä avaan palvelimen lähettämän vastauksen selaimen *alert*-ikkunaan:
  ```
  alert(myRequest.responseText);
  ```

4 Dataformaatit tiedonsiirrossa

Ajax ei ole valikoiva sen kanssa käytettävän palvelinpuolen ohjelmointikielen suhteen, mutta palvelimen XHR-objektille palauttaman vastauksen muodon suhteen on sen sijaan tehtävä valinta muutaman käytössä olevan dataformaatin välillä. Palvelimen on lähetettävä vastaus XML-, JSON-, HTML tai teksti-muodossa. (Keith 2007, 68)

Kuten edellisessä luvussa todettiin, XHR-objekti tarjoaa kaksi tapaa päästä käsiksi palvelimen lähettämään vastaukseen – *responseText* sekä *responseXML*. Ensimmäinen palauttaa palvelimen lähettämän vastauksen tekstimuodossa, merkkijonona, mutta mikäli palvelimen lähettämä vastaus on XML-olio, voidaan se lukea vastaavasti *responseXML*:n kautta. (Asleson & Schutta 2006, 29)

4.1 Tekstimuotoiset vastaukset

Yksinkertaisimmillaan palvelimen XHR-objektille palauttama vastaus voi olla jokin merkkijono ilman minkäänlaista semanttista muotoilua – siis esimerkiksi yksi merkki, sana tai vaikka kokonainen tekstikappale.

Perinteisimmillään Ajaxia on sovellettu tiedon reaaliaikaiseen validointiin. Esimerkiksi voidaan ottaa käyttäjän lomakekenttään syöttämä sähköpostiosoite tai jokin muu tieto ja välittää se palvelimelle. Palvelimella tieto validoidaan ja tuloksena XHR-objektille palautetaan esimerkiksi *true* tai *false*, jonka mukaan sivulla voidaan päivittää huomautus validoinnin läpäisseestä tai reputtaneesta syötteestä.

4.2 XML

XML (eXtensible Markup Language) on W3C:n kehittämä yleiskäyttöinen merkintäkieli, jota käytetään pääasiallisesti tietojärjestelmien välisen tiedonsiirron dataformaattina internetissä. XML:n syntaksi on ennalta määritelty, mutta se sallii käyttäjän itse määritellä käytettävän sanaston. Tästä syystä se soveltuu ihanteellisesti myös kustomoitujen Ajax-sovellusten dataformaatiksi. (Wikipedia: XML) Se on yleisesti myös useiden API- ja Web service -palveluja tarjoavien tahojen käytössä, mikä tekee siitä suositun dataformaatin erilaisten mashupien toteutuksessa.

XML-dokumentin parsiminen JavaScriptillä ei edellytä minkään uuden kielen opiskelua, edellyttäen, että DOM-metodit ovat entuudestaan tuttuja. DOM-skriptaamisen käyttö ensin XML-dokumentin läpikäymiseen ja sitten kerätyn datan saattaminen HTML-muotoon tuottaa helposti kuitenkin suuren määrän koodia, kun jokainen elementti ja solmu (engl. node) on luotava ja lisättävä erikseen. Erityisesti isoja datamääriä liikuteltaessa tämä saattaa koitua ongelmalliseksi. Vaihtoehto DOM-metodien käyttämiselle XML-dokumentin HTML-muunnokseen on XSLT, mutta sen selaintuki on vajavainen. (Keith 2007, 77)

4.2.1 Esimerkki XML-muotoisen vastauksen käsittelystä

Otetaan esimerkki, jossa palvelin on palauttanut XHR-objektille seuraavanlaisen XML-dokumentin:

```
<?xml version=”1.0” encoding=”utf-8”?>
<message>
  <author>Sami Harju</author>
  <body>Hello world!</body>
  <date>2008-10-25</date>
</message>
```

XML-dokumenttia voidaan käydä läpi JavaScriptillä hyödyntäen täysin samoja DOM-metodeja, kuin HTML-dokumentin kanssa. Kun palvelimelta palautettavan XML-dokumentin rakenne on tiedossa, on sen selaaminen suhteellisen suoraviivainen prosessi käyttäen apuna metodeja kuten *getElementsByTagName* ja ominaisuuksia kuten *nodeValue*. (Keith 2007, 77; W3Schools, XMLHttpRequest)

Muokataan aiemmin tehtyä *onreadystatechange*-tapahtumankäsittelijään asetettua *parseResponse*-funktiota siten, että se vastaanottaa palvelimelta tulleen XML-olion ja parsii siitä DOM-mallia hyödyntäen viestin lähettäjän nimen dokumentin `<author>`-elementistä ja asettaa sen *author* nimiseen JavaScript-muuttujaan ja vastaavasti viestin `<body>`-elementistä *msg*-muuttujaan.

```
function parseResponse() {
  if(myRequest.readyState == 4) {
    if(myRequest.status == 200 || myRequest.status == 304) {
      var myXML = myRequest.responseXML;
      var author = myXML.getElementsByTagName('author')
[0].firstChild.nodeValue;
```

```
      var msg = myXML.getElementsByTagName('msg')
[0].firstChild.nodevalue;
    }
  }
}
```

Kun data on saatu parsittua XML-dokumentista, se halutaan useimmiten esittää käyttäjälle tavalla tai toisella. Seuraavassa lyhyessä esimerkissä yllä haettu viestin kirjoittajan nimi lisätään otsikoksi HTML-dokumentin `<div>`-elementtiin, joka on nimetty *id*:llä *message*, ja viesti otsikon alle `<p>`-elementin sisään. Tämä onnistuu lisäämällä seuraavat rivit *parseResponse*-funktioon yllä luotujen uusien rivien jälkeen:

```
var message = document.getElementById('message');
var header = document.createElement('h1');
var headerText = document.createTextNode(author);
header.appendChild(headerText);
var paragraph = document.createElement('p');
var paragraphText = document.createTextNode('msg');
paragraph.appendChild(paragraphText);
message.appendChild(header);
message.appendChild(paragraph);
```

Lisäksi esimerkiksi *setAttribute('attribuutti', 'arvo')* -DOM-metodin avulla voitaisiin lisätä elementeille attribuutteja, joilla niille voitaisiin antaa vaikkapa *id*- tai *class*-määreitä ja siten kohdistaa niihin CSS-sääntöjä ja niin edelleen.

4.3 JSON

JSON (JavaScript Object Notation) on kevyt tiedonsiirtoformaatti, joka kehitettiin 2000-luvun alussa Douglas Crockfordin toimesta, ja joka perustuu JavaScript-objektien merkintätapaan. JSON:ia käytetään yleisesti JavaScriptin kanssa, mutta sitä pidetään kuitenkin ohjelmointikieliriippumattomana dataformaattina. Useat ohjelmointikielet tarjoavat jo valmiit työkalut JSON:in tuottamiseen ja jäsentämiseen; täydellinen lista JSON:ia tukevista kielistä löytyy osoitteesta http://www.json.org. Valtaosa sen sovelluskohteista löytyy nimenomaan Ajax-sovellusten piiristä, jossa se on niittänyt laajamittaista suosiota vaihtoehtona XML:lle. (Crockford, 2006; Wikipedia, JSON)

JSON-syntaksi koostuu siis avain-arvo pareista, jotka ovat erotettu pilkuin ja suljettu kaarisulkeisiin. JSON voi pitää sisällään dataa merkkijonona, numeroina, boolean-

arvoina (*true* / *false*), objekteina sekä taulukkoina tai *null*-arvoina. Aiemmin XML-esimerkissä käytetty data voidaan merkitä JSON-muodossa näin:

```
{
  ”message”: {
    ”author”: ”Sami Harju”,
    “body”: ”Hello world!”,
    ”date”: ”2008-10-25”
  }
}
```

Toisin kuin XML, JSON ei vaadi JavaScriptilta varsinaista tulkitsemisprosessia, sillä se on jo itsessään syntaksiltaan JavaScriptia. Sen parsimiseen ei siis tarvita DOM-metodeja, vaan se voidaan muuttaa käytettäväksi tietorakenteeksi yksinkertaisesti JavaScriptin sisäänrakennettua *eval*-funktiota tai turvallisemmin esimerkiksi JSON.org:in tarjoamaa JSON-parseria hyödyntäen. (Keith 2007, 78; Crockford, 2006) Parsitun datan HTML-muotoon saattaminen on tämän jälkeen samanlainen prosessi kuin XML:n kohdallakin.

4.3.1 eval

eval on JavaScriptin sisäänrakennettu globaali funktio, joka ei ole osa mitään objektia. Se ottaa parametrinä sisäänsä merkkijonon, jonka se evalvoi eli laskee tai suorittaa JavaScriptinä. Sen avulla voidaan siis kääntää esimerkiksi palvelimelta saatu JSON-muotoinen merkkijono JavaScript-objekteiksi. (Mozilla Developer Center)

Tapauksessa, jossa palvelin palauttaisi XHR-objektille vastauksen JSON-merkkijonona, se olisi luettavissa XHR-objektin *responseText*-ominaisuuden kautta. Vastaus täytyy aina evalvoida ennen kuin dataan päästäisiin käsiksi esimerkiksi seuraavalla tavalla:

```
var myJSON = eval('('+myRequest.responseText+')');
var author = myJSON.message.author;
var body = myJSON.message.body;
var date = myJSON.message.date;
```

eval-funktion käyttöön liittyy kuitenkin tietoturvariskejä, mikäli käsiteltävä data ja koko JavaScript-ympäristö eivät ole yhden luotetun lähteen takana – ja tietysti myös tilanteessa, jossa palvelin kaapataan. Epäluotettavasta lähteestä saatu data voi sisältää

esimerkiksi vahingollista JavaScript-koodia, joka suoritetaan evalvoinnin yhteydessä automaattisesti, ellei dataa ensin validoida tavalla tai toisella.

4.3.2 JSON-parserit

Tietoturvallisuussyistä on yleisesti suositeltavaa käyttää JSON-merkkijonon käsittelemiseen tarkoitettua parseria, eli jäsentäjää, joka jäsentää ainoastaan JSON-merkkijonon ja hylkää mahdollisen vahingollisen sisällön kuten erilaiset metodit ja niiden kutsut.

JSON.org tarjoaa ladattavaksi open source -parserin käyttöohjeineen osoitteessa http://www.json.org/js.html. Sen avulla JSON-merkkijonon jäsentäminen JavaScript-objektiksi tapahtuu *json2.js*-tiedoston ollessa linkitettynä dokumenttiin yksinkertaisimmillaan välittämällä merkkijono *JSON*-objektin *parse*-funktion parametriksi:

```
var myJSON = JSON.parse(myRequest.responseText);
```

4.4 HTML

Kun vastaus palvelimelta lähetetään XML- tai JSON-muodossa, se on muutettava HTML-muotoon ennen kuin se voidaan näyttää selaimessa. Sen sijaan, jos palvelin palauttaa datan valmiiksi HTML-muotoiltuna, se voidaan liittää päivitettävään dokumenttiin sellaisenaan. HTML on puhtaasti tekstuaalista dataa, joten se on luettavissa XHR-objekti *responseText*-ominaisuuden kautta.

Yksinkertaisin tapa liittää palautettu HTML osaksi päivitettävää dokumenttia on hyödyntää JavaScriptin *innerHTML*-ominaisuutta. Se on alun perin Microsoftin Internet Explorerille kehittämä epästandardi ominaisuus, joka on vakiintunut de facto-standardin asemaan sen poikkeuksellisen laajan selaintuen vuoksi. Muut selaimet ovat ottaneet sen käyttöön ilman W3C:n suositusta, sillä se on todella käytännöllinen tietyissä tilanteissa. Siinä, missä DOM-metodeilla voidaan luoda tarkasti elementtejä solmu kerrallaan, käyttää *innerHTML* raakaa voimaa. Sitä käyttäessä kaikki elementin aiempi sisältö korvataan elementin *innerHTML*:ään asetetulla sisällöllä. Toisin kuin DOM-metodeja hyödyntäen, on *innerHTML*:n kautta mahdollista rikkoa dokumentin DOM-malli syöttämällä elementtiin virheellisesti muotoiltua HTML:ää. Huolimattomasti käytettynä

se voi siis olla dokumentin rakenteen kannalta vahingollista, mutta toisaalta se voi myös säästää huomattavan määrän aikaa ja koodirivejä.

Edellisten esimerkkien mukainen päivitys dokumenttiin tapahtuisi seuraavalla tavalla, mikäli palvelin palauttaisi halutut elementit valmiiksi HTML-merkkijonona XHR-objektille.

```
function parseResponse() {
  if(myRequest.readyState == 4) {
    if(myRequest.status == 200 || myRequest.status == 304) {
      var message = document.getElementById('message');
      message.innerHTML = myRequest.responseText;
    }
  }
}
```

HTML-muotoisen vastauksen palauttaminen sopii hyvin tilainteisiin, joissa päivitetään vain yksi osa dokumenttia ja tiedetään, että pyydetyn palvelinresurssin ei tarvitse palauttaa samaa dataa kuin yhteen käyttökohteeseen. Useita dokumentin eri osia päivitettäessä sen sijaan on syytä käyttää XML:ää tai JSON:ia, koska niiden sisältämä data voidaan parsia ja muotoilla vapaasti.

5 Suunnittelu ja implementointi

Ajaxin sovelluskohteet ulottuvat sivuston käytettävyyttä parantavista pienistä ratkaisuista perinteisissä web-käyttöliittymissä aina täysverisiin sovelluksiin, joiden toiminnallisuus ja luonne ovat lähellä perinteisiä työpöytäsovelluksia. Tämä luonnollisesti asettaa täysin erilaisia haasteita suunnittelun näkökulmasta. Täysveristen Ajax-sovellusten suunnitteluun pätevät pääosin samat säännöt kuin perinteiseen sovellussuunnitteluunkin, joten en keskity niihin. Sen sijaan nostan esiin joitakin suunnittelullisia seikkoja, jotka koskevat erityisesti Ajaxia.

Kuten minkä tahansa pinnalla olevan uuden tekniikan kanssa, myös Ajaxin kohdalla tulisi välttää hype-ohjautuvaa suunnittelua. Se voi parantaa sivuston käytettävyyttä monin tavoin, mutta väärään kohtaan sovellettuna lopputulos voi olla päinvastainen. Kehittäjän tulisi pitää sivuston käyttäjät ja käytettävyys aina ensimmäisenä mielessä valitessaan hyödynnettäviä tekniikoita. Ajaxia ei tulisi käyttää vain sen itsensä vuoksi – käyttäjiä kiinnostaa jouheva käyttökokemus, ei sen saavuttamiseen käytetty tekniikka. (Asleson & Schutta 2006,22)

Erilaisiin ratkaisumalleihin on hyvä tutustua etukäteen muiden kehittäjien rakentamilla sivuilla, ja lisäksi Ajax-suunnittelumalleja on kehitetty ja koottu muun muassa AjaxPatterns-sivustolle (http://www.ajaxpatterns.org). Suunnittelumalleihin ja Ajaxin ympärille kehittyviin konventioihin tukeutuminen auttaa usein hyvin käytettävän sivuston suunnittelussa.

Web-maailmassa on sen vuosien olemassa olon aikana ehditty tottua synkroniseen tiedonsiirtoon ja kokonaisiin sivupäivityksiin perustoimintojen yhteydessä. Ajax lisää sivuston käyttöön asynkronisen ulottuvuuden, joka saattaa yllättää käyttäjän. Koska erilaiset selaimen tavat indikoida sivun latausta, kuten oikeassa yläkulmassa animoitu logo tms., puuttuvat Ajax-pyyntöjen yhteydestä kokonaan, paras tapa ottaa käyttäjä huomioon, on antaa hänelle itse palautetta suoritettavien toimintojen yhteydessä. Jo jonkinasteiseksi konventioksi on kehittynyt erilaisten latausindikaattorianimaatioiden käyttö ja päivittyneen sisällön korostaminen esimerkiksi *Yellow Fade Technique* (YFT, Basecamp) tai *Fade Anything Technique* (FAT, Adam Michela) -tekniikoita hyödyntäen. Näiden tekniikoiden implementointi Ajax-pyynnön yhteyteen on helppoa ja melko vaivatonta – pyynnön alussa luodaan elementti, joka näyttää latausanimaation

(tavallisesti GIF-kuva) ja vastaavasti pyynnön päättyessä, se poistetaan ja päivittynyt sisältö liitetään sivulle samalla muuttaen sen taustaväri korosteväristä asteittain normaaliin taustaan häivyttäen joko itse koodatulla skriptillä tai yllä mainittuja *YFT*/*FAT*-tekniikoita hyödyntäen. Latausindikaattoreita ja fade-tekniikoita käyttäessä on kuitenkin hyvä harkita, missä niitä tarvitaan ja missä pärjätään ilmankin, sillä niiden liiallinen käyttö sekavoittaa käyttökokemusta.

Puuttuva latauksen indikointi ei ole ainoa Ajaxin potentiaalisista vaikutuksista selaimen käyttäytymiseen. Monet muutkin selaimiin rakennetuista toiminnoista perustuvat perinteiseen koko sivun päivittävään selaamisen malliin – näistä Ajaxin kannalta ongelmallisimpia ovat käyttäjän mahdollisuus tallettaa sivu kirjanmerkkeihin ja selaimen ylläpitämä sivuhistoria, johon pohjautuu myös *edellinen*- ja *seuraava*-painikkeiden käyttö. Nämä ominaisuudet perustuvat siihen, että jokaisella pyydetyllä resurssilla on oma, uniikki URL-osoitteensa. Ajax kuitenkin päivittää pyydetyn resurssin auki olevalle sivulle muuttamatta URL:ia, joten tämä malli rikkoutuu, eikä sivuhistoriaa luoda tai kirjanmerkkejä voida päivitetyllä sisällöllä tallentaa. Ongelmaan on kehitetty erilaisia ratkaisuja, mutta ainakaan toistaiseksi niistä ei yksikään ole noussut de facto -standardin asemaan, tai korjannut ongelmaa tarpeeksi siististi, joten en nosta niitä esiin tämän enempää. Sen sijaan, että tilanne yritettäisiin korjata kiertoteitse, voidaankin miettiä sivun toiminnallisuutta suunniteltaessa, onko Ajax sittenkään oikea vaihtoehto. Jos sivun sisältöä päivitetään niin paljon, että käyttäjä uskoo siirtyneensä kokonaan uudelle sivulle, tehdään luultavasti jo liikaa. Sen sijaan, jos Ajaxia käytetään päivittämään vain pieniä osia sivusta, ei ongelmaa sivuhistorian tai kirjanmerkkauksen kanssa luultavasti synny. (Keith 2007, 130 & 136)

5.1 Ajax ja saavutettavuus

Sivuston saavutettavuudesta näkee Ajaxin yhteydessä puhuttavan kahdesta eri asiasta: selaimista, jotka eivät ole JavaScript-kykeneviä syystä tai toisesta ja näkökykynsä puolesta rajoittuneesta, ruudunlukijaa käyttävästä kohderyhmästä. Nämä ovat kaksi suurinta haastetta Ajaxin kannalta.

James Edwards on tehnyt tutkimusta siitä kuinka ruudunlukijat tulkitsevat DOM-mallia dynaamisesti päivittäviä skriptejä. Tutkimustulokset näyttävät suoraan sanoen melko synkiltä, eikä niiden valossa Ajaxia voida kutsua todellisesti saavutettavaksi tekniikaksi.

Edwards on julkaissut tutkimuksistaan artikkelin *AJAX and Screenreaders: When Can it Work?* (2006), joka on luettavissa webissä osoitteessa http://www.sitepoint.com/article/ajax-screenreaders-work/. Joitakin lähestymistapoja saavutettavuutta parantamaan on jo kehitetty, mutta mikään niistä ei kata kaikkia markkinoilla olevia ruudunlukijoita. Elämme kuitenkin vielä Ajaxin alkuvuosia, joten saavutettavuutta parantavat ratkaisut voivat odottaa löytämistään aivan nurkan takana – niin avustavia välineitä kehittävien tahojen kuin web-kehittäjienkin toimesta.

Toinen suuri haaste Ajaxia hyödyntävän sivuston suunnittelussa on siis sen täydellinen riippuvaisuus JavaScriptin saatavilla olosta käyttäjän selaimessa. Ilman JavaScriptiä ei ole Ajaxia. Sivuston tärkein osa on oletettavasti sen sisältö. Sisällön saavutettavuus pitäisi pystyä takaamaan myös käyttäjille, jotka selaavat internetiä ilman JavaScriptiä. Yksi vaihtoehto on luoda vaihtoehtoinen, yksinkertaisempi, JavaScriptitön versio sivustosta. On kuitenkin selvää, että se moninkertaistaa sivuston rakentamiseen käytettävän ajan ja merkittävästi vaikeuttaa sivuston ylläpitoa. Lisäksi lähtökohtaisesti tämä asettaa sivuston eri versiot eri arvoasemaan täten arvottaen myös sivuston käyttäjiä. Ihannetapauksessa sivuston tulisi sopeutua selaimen vaatimuksiin ja kykyihin. (Keith 2007, 94)

5.1.1 Progressive enhancement

Progressive enhancement (tästä eteenpäin PE) on web-suunnittelun strategia, joka painottaa saavutettavuutta, semanttista merkintää sekä ulkoisten tyylitiedostojen ja skriptausteknologioiden käyttöä. Se käyttää sivuston web-teknologioita kerroksellisesti mahdollistaen perussisällön ja -toiminnallisuuden saavutettavuuden käytettäen mitä tahansa selainta tai internet-yhteyttä, samalla kuitenkin tarjoten parannellun version sivustosta niille, joiden käytössä on esimerkiksi laajakaistayhteys tai edistyksellisempi selain. (Wikipedia, Progressive Enhancement) Ajaxin kannalta PE-prinsiippien noudattaminen edellyttää toiminnallisuuden suunnittelua projektin alusta lähtien ja sen implementointia viimeisenä. Ensin siis luodaan perinteiseen, sivupohjaiseen interaktiomalliin perustuva toiminnallisuus ja lopuksi sen päälle implementoidaan haluttu Ajax-kerros. Täten vaihtoehtoista JavaScriptitöntä versiota sivusta ei tarvitse tehdä erikseen – se on jo olemassa.

Käytännössä PE:n prinsiippien osittainen noudattaminen on suurelle osalle kehittäjistä entuudestaan tuttua – sisällön ja esitystavan erottaminen hyödyntäen semanttista XHTML:ää ja ulkoisia CSS-tyylitiedostoja. Kun Ajax astuu mukaan kokonaisuuteen, lisätään siis edellä mainittujen sisältö- ja esitystapa-kerrosten päälle kolmas, toiminnallisuus-kerros. Sivuston perustoiminnallisuutta ja käyttökokemusta parannetaan ulkoisista JavaScript-tiedostoista ladattavalla koodilla, joka kytketään eri elementteihin olettamatta JavaScriptin olevan päällä. Täten ilman JavaScriptiä sivuston perustoiminnallisuus on edelleen saavutettavissa. Tätä lähestymistapaa kutsutaan unobtrusive JavaScriptiksi, tunkeilemattomaksi JavaScriptiksi

5.1.1.1 Unobtrusive JavaScript

Kuten sivuston tyylimääritykset, myös sen toiminnallisuus kannattaa pitää erillään sisällöstä. CSS-tyylisäännöt on totuttu kirjoittamaan ulkoisiin tyylitiedostoihin ja sisällyttämään sivuun hyödyntäen HTML:n `<link>`-tagia. Myös JavaScript-koodi voidaan kirjoittaa ulkoiseen *.js*-tiedostoon ja sitten linkittää HTML-dokumenttiin `<script>`-tagilla.

Toiminnallisuus liitetään eri elementteihin käyttäen tapahtumienkäsittelijöitä. Ne on mahdollista kirjoittaa suoraan HTML-elementin yhteyteen, mutta se tuottaa HTML:ään lopputuloksen kannalta turhia koodirivejä, jotka eivät suoranaisesti liity dokumentin sisältöön. Sen sijaan on hyvä tapa käyttää CSS-valitsimia, joilla halutut elementit poimitaan dokumentista ja niihin liitetään haluttu toiminnallisuus DOM-mallia hyödyntäen. Yksi ongelma tässä kuitenkin on – DOM:in täytyy olla ladattu, ennen kuin elementtiin päästään DOM-metodeilla käsiksi. Tämä voidaan ratkaista joko asettamalla `<script>`-tagi juuri ennen dokumentin sisältöalueen sulkevaa `</body>`-tagia tai asettaa haluttu toiminto JavaScriptin *window*-objektin *onload*-tapahtumankäsittelijään, joka suoritetaan heti sivun latauduttua.

Seuraavassa yksinkertaisessa esimerkissä on tavallinen HTML-linkki, joka nimetty *id*:llä *link*.

```
<a href=”sivu.html” id=”link”>Klikkaa minua!</a>
```

Linkki poimitaan DOM-metodilla sille asetetun *id*:n mukaan ja sille lisätään jokin toiminnallisuus. Tämä tapahtuu, kun dokumentti on kokonaisuudessaan ladattu.

Huomion arvoista on se, että linkin *href*-attribuutti ohjaisi käyttäjän halutulle sivulle siinä tapauksessa, jos JavaScript ei olisi käytössä.

```
window.onload = function() {
            var myLink = document.getElementById('link');
            myLink.onclick = doSomething();
}
```

Usein sivun lataamisen yhteydessä halutaan käynnistää useita toimintoja. JavaScriptin *window*-objektille ei kuitenkaan oletusarvoisesti voida asettaa kuin yksi funktio. Ratkaisumalli tähän ongelmaan löytyy muun muassa Simon Willisonin kirjoittamasta *addLoadEvent*-funktiosta (http://www.simonwillison.net) sekä useista JavaScript-kirjastoista. JavaScript-kirjastoja ja -työkaluja käsittelen omassa luvussaan tarkemmin.

6 JavaScript-kirjastot

JavaScript-kirjasto on kokoelma työkaluja ja oikopolkuja JavaScript-kehityksessä eteen tulevien ongelmien ratkaisemiseen. Sen tarkoitus on mahdollistaa kehittäjien keskittymisen sovelluksensa kannalta keskeisten toiminnallisuuksien koodaamiseen sen sijaan, että aikaa kuluisi aina kehitystyötä aloittaessa toistuvien perustoimintojen määrittelyyn. Lisäksi työkalut ja oikopolut nopeuttavat kehitystä yleisesti ottaen yksinkertaisesti vähentämällä erilaisten toimintojen, tai esimerkiksi HTML-elementtien, luomiseen tarvittavien koodirivien määrää. (Holdener III 2008, 97) Kehittäjät ovat rakentaneet omia kirjastojaan törmätessään toistuvasti samankaltaisiin tilanteisiin ja ongelmiin sovelluksia kirjoittaessaan. Ongelmiin löytyneet ratkaisut ovat muotoiltu geneerisiksi funktioiksi ja tallennettu myöhempää käyttöä varten täten hiljalleen kasvattaen syntyvän kirjaston sisältöä. Laajempia kirjastoja on julkaistu myös muiden kehittäjien käyttöön. Näistä suositumpia ovat Prototype (http://www.prototypejs.org) ja sen päälle rakennettu animaatio- ja UI-komponentti -kirjasto script.aculo.us (http://script.aculo.us), The Dojo Toolkit (http://www.dojotoolkit.org) sekä jQuery (http://www.jquery.com), jotka kaikki ovat vapaan lisenssin alaisia open source -kirjastoja. Vaikka valmiin kirjaston käyttö vaikuttaa houkuttelevalta vaihtoehdolta helppoon JavaScript-kehittämiseen, on kehittäjän kuitenkin hyvä hallita JavaScript perusteellisesti ennen kirjaston käyttöönottoa, jotta hän voi tarvittaessa jäljittää sen käytön yhteydessä tapahtuneet virheet ja korjata ne, eli ymmärtää kuinka toiminnot ovat rakennettu.

Tavallisesti kirjastoista löytyy työkaluja DOM-skriptaamiseen, tapahtumankäsittelyyn, elementtien animointiin sekä Ajax-hallintaan. Edellä mainittuja yhdistävä tekijä on niiden tarve toisistaan poikkeaville ratkaisuille eri selaimia varten. Täten on ollut järkevää koota yhteen funktioita ja objekteja, jotka hoitavat tarvittavat tarkistukset käyttäjän selaimen suhteen ja valitsevat sitten käytettävän ratkaisun, sen sijaan, että esimerkiksi XHR-objektia tai jokaista DOM-tapahtumankäsittelijää alustettaessa koodiin jouduttaisiin kirjoittamaan useita ylimääräisiä rivejä selaintyypin testaamiseksi ja selaintuen varmistamiseksi. Useiden JavaScript-kirjastojen suurin vahvuus piileekin niiden huolellisesti testatussa selaintuessa. Kirjastot listaavat tavallisesti www-sivullaan tukemansa selaimet, mikä on syytä tarkistaa ennen kirjaston käyttöönottoa. Monet kirjastoista ovat vielä vahvasti kehityksen alla, eikä niiden käytöllä ole missään

tapauksessa perusteltua sulkea selaimia pois vajavaisen tuen vuoksi. (Keith 2007, 189-190)

Käytettävää kirjastoa valitessa on mietittävä tarkkaan omat tarpeet tarjolla olevien työkalujen suhteen pitäen mielessä sen, että JavaScript suoritetaan aina käyttäjän selaimessa, joten sivulle linkitetty kirjasto on pakko ladata muun sisällön yhteydessä. Jotkut kirjastot ovat jaettu moduuleihin, jolloin on mahdollista ottaa vain tarvittavat komponentit käyttöön ja näin ollen säästää ladattavissa kilotavuissa. Useat kirjastot sen sijaan tulevat yhdessä JavaScript-tiedostossa, jolloin käyttäjän on ladattava se kokonaisuudessaan, vaikka kehittäjä olisi hyödyntänyt ainoastaan yhtä sen osista. Tietyn kirjaston käyttöönotto ei siis välttämättä ole perusteltua esimerkiksi tilanteessa, jossa tarvitaan vain nopea ratkaisu XHR-pyyntöjen käsittelyyn, mutta joudutaan samalla lataamaan koko joukko animointiin ja DOM:in manipulointiin tarkoitettuja metodeja.

Kirjastojen dokumentaation taso on vaihteleva. Monet niistä tarjoavat useita tehokkaita työkaluja kehittäjille, mutta ilman kunnollista ohjeistusta ne jäävät helposti käyttämättä. Useimmat tarjoavat jonkinlaisen dokumentaation käyttäjilleen, mutta myös sen tyyli vaihtelee kirjastokohtaisesti. Joidenkin kirjastojen dokumentaatio on lähestymistavaltaan ylettömän tekninen, eikä siten kovin helppolukuinen etenkään aloittelevalle kehittäjälle. Toisaalta monet kirjastot nauttivat aktiivisen käyttäjäkuntansa tuesta käytännönläheisten, kirjaston ominaisuuksia käsittelevien blogi- ja foorumikirjoitusten muodossa. (Keith 2007, 193)

7 Case-esimerkki: Audionce

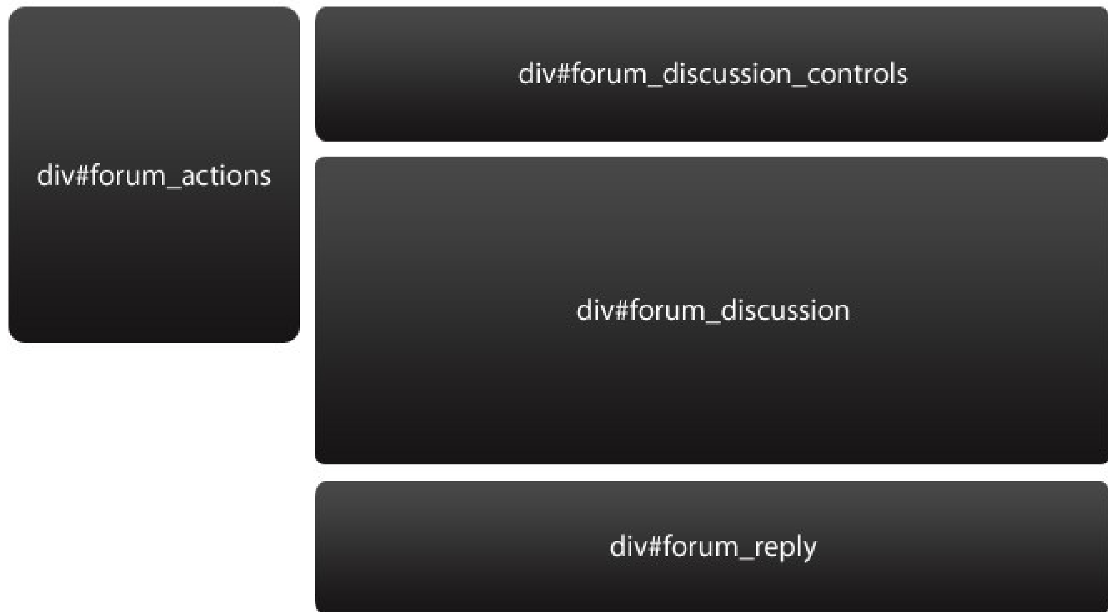
Valitsin käytännön case-esimerkiksi kehitteillä olevan vapaan musiikin jakeluun perustuvan sosiaalisen verkkopalvelun Audioncen. Sivustolla hyödynnetään Ajaxia monissa toiminnoissa, mutta tarkemmin tarkastelen sen keskustelupalstaa ja uuden viestin lisäämistä keskusteluun. Kyseinen toiminto on toteutettu hyväksikäyttäen Prototype-kirjastoa Ajax-pyyntöjen käsittelyyn ja DOM-skriptaamiseen, XML:ää tiedonsiirtoformaattina sekä progressive enhancement -prinsiippejä toiminnallisuuskerroksen implementointiin.

Sivuston eri toimintoja on haluttu höystää Ajaxilla, jotta saavutettaisiin mahdollisimman jouheva käyttökokemus ja vältyttäisiin tarpeettomilta sivun uudelleen latauksilta tilanteissa, jossa ne eivät ole välttämättömiä. Foorumin toimintoihin uuden viestin lisäämisestä, viestin muokkaamiseen ja poistamiseen Ajax soveltuu hyvin. Esimerkiksi uusi lähetetty viesti voidaan esittää kirjoittajalle heti, kun hän on hyväksynyt sen nappia painamalla, samalla korostaen päivittynyttä sisältöä sopivalla visuaalisella vihjeellä. Tämä toiminnallisuus rakennetaan perinteisen, koko sivun päivityksiin perustuvan järjestelmän päälle, jolloin myös ilman JavaScriptiä surffaavat käyttäjät voivat kirjoittaa foorumille.

Kehittäjän kannalta tämä tarkoittaa yhden ylimääräisen kerroksen suunnittelua ja toteuttamista. JavaScriptiin perustuva toiminnallisuuskerros ladataan sivulle olettamatta käyttäjän selaimen tukevan sitä lainkaan. Koska pohjalle on jo rakennettu toimiva palvelinpuolen ratkaisu ja peruskäyttöliittymäelementit XHTML-muodossa, on Ajaxiin pohjaavan toiminnallisuuden lisääminen keskustelupalstalle verrattain suoraviivainen prosessi. Lopputulos vaatii kuitenkin enemmän työtä ja tuottaa suuremman määrän koodirivejä, mutta jos Ajax suunnitellaan aluksi ja toteutetaan lopuksi, ei prosessi kokonaisuudessaan ole merkittävän monimutkainen.

7.1 Perustoiminnallisuuden suunnittelu ja toteutus

Prosessi alkaa perinteisen foorumitoiminnallisuuden suunnittelulla. Sivun eri osat jaetaan moduuleihin, jotka kukin hoitavat yhden osan tarvittavasta toiminnallisuudesta tai sisällön esittämisestä. Sivun jakaminen moduuleihin helpottaa sen käsittelyä DOM-mallin avulla. Ulkoasumääritykset kootaan ulkoiseen tyylitiedostoon.



Kaaviokuva foorumin HTML-rakenteesta

- `div#forum_actions`: perustoiminnallisuudet ja -navigaatio
- `div#forum_discussion_controls`: otsikkotiedot sekä (admin-)työkalut
- `div#forum_reply`: vastaus-lomake, eli HTML-`<form>`
- `div#forum_discussion`: keskustelun postit omissa *id*:llä merkityissä `<div>`:eissään:

Kaaviokuva yksittäisen viestin HTML-rakenteesta

- `div#message_n`: isäntäelementti viestille
- `div.forum_comment_writer`: kirjoittajan tiedot (nimimerkki, kuva ym.)
- `div.forum_comment_text`: viestin sisältö ja päiväys
- `div.forum_comment_tools`: viestiä koskevat työkalut (lainaa, muokkaa, poista)

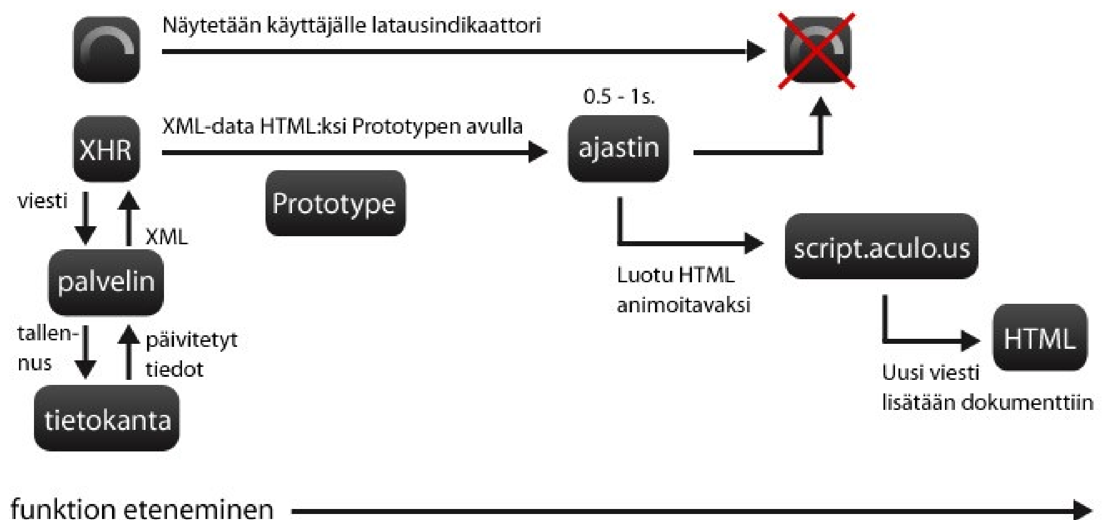
Koska Ajaxin kannalta ei ole merkityksellistä, mitä ohjelmointikieltä palvelimella käytetään, en esittele siellä tapahtuvaa toiminnallisuutta tarkemmin – oleellista on se, että tässä vaiheessa palvelimelle luodaan pyydettyjä toimintoja käsittelevät moduulit, jotka hoitavat tietokantatoiminnot, virheenkäsittelyn ja niin edelleen. Tämän jälkeen foorumi on perustoiminnallisuuksiltaan valmis, ja siihen voidaan liittää JavaScript-toiminnallisuuskerros.

7.2 Toiminnallisuuskerroksen implementointi

PE-periaatteen mukaisesti JavaScript-toiminnallisuus kootaan ulkoiseen tiedostoon tai tiedostoihin, jotka linkitetään XHTML-runkoon. Dokumentin latauduttua suoritetaan funktio, jonka kutsu siis sijoitetaan ennen dokumentin sisältöalueen sulkevaa `</body>`-tagia, ja joka poimii DOM:ista halutut linkit ja painikkeet uudelleen alustettaviksi. Painikkeisiin liitetään tapahtumankäsittelijät, jotka ohittavat niiden oletuksellisen toiminnallisuuden. Samalla uuden postin lähettävän lomakkeen onsubmit-tapahtumankäsittelijään liitetään toiminto, joka ohjaa palvelinliikenteen kulkemaan XHR-objektin kautta ja lopuksi liittää uuden viestin XHTML-dokumenttiin.

Käytännössä kaikki foorumin toiminnot ovat omissa JavaScript-funktioissaan, josta niitä tarvittaessa kutsutaan. Yllämainittu, uuden viestin lähettämiseen käytetty funktio on niistä monimutkaisin. Se poimii lomakkeesta käyttäjän syöttämän viestin ja välittää sen sekä tarvittavat tunnisteet, kuten keskusteluketjun ID:n, palvelimelle XHR-objektin kautta. Samanaikaisesti se liittää XHTML-dokumenttiin latausindikaattorin ja passivoi lähetyslomakkeen kentät, jotta käyttäjä tietää pyytämäänsä toimintoa käsiteltävän. XHR-objektin kutsuma palvelinskripti tallentaa uuden viestin tietokantaan ja palauttaa sitten viestin sekä päivitetyt käyttäjätiedot XML-muodossa takaisin XHR-objektille. Sen *responseXML*-ominaisuudesta XML-dokumentti luetaan JavaScript-muuttujaan ja jäsennetään Prototype-kirjaston DOM-metodeja hyväksi käyttäen XHTML-muotoon.

Kun uusi viesti on valmis, alustetaan 0.5 – 1 sekuntia kestävä viiveajastin, jotta dokumenttiin liitetty latausindikaattori näkyy käyttäjälle tarpeeksi kauan riippumatta palvelimen vastausnopeudesta. Lopuksi ajastin laukaisee funktion, joka poistaa latausindikaattorin ja lisää luodun viestin XHTML-dokumenttiin. Ennen kuin viesti näytetään käyttäjälle, se animoidaan esiin script.aculo.us -kirjastoa hyödyntäen, täten selkeästi esittäen uuden viestin liittämisen dokumenttiin.



Lomakkeen onsubmit-tapahtumankäsittelijään lisättävän funktion kulku

7.3 Johtopäätökset

Audioncen foorumin kohdalla Ajaxin lisääminen perustoiminnallisuuden päälle loi näyttävän ja jouhevan käyttökokemuksen säilyttäen tuen JavaScriptittömillekin selaimille. Ajaxin tuoma parannus käyttökokemukseen näkyy erityisesti silloin, kun palvelin on kuormitettu, ja viestin tallentamineen tietokantaan kestää tavallista kauemmin – sivun toiminta ei näytä pysähtyvän missään vaiheessa. Lisäksi Ajax-pyynnön yhteyteen on suhteellisen vaivatonta rakentaa erilaisia ratkaisuja virheenhallintaan – esimerkiksi erilaiset virheilmoitukset, mahdollisuus yrittää samaa toimintoa uudelleen saumattomasti, aikarajoite suoritukselle ja niin edelleen.

Koska kyseessä ei ole kaupallinen projekti, käytettyjen työtuntien ja lopputuloksen hyötysuhdetta on vaikea punnita – usein on kuitenkin syytä harkita jo projektin suunnitteluvaiheessa onko Ajaxin tuoma parannus käytettävyyteen sen suunnitteluun ja toteuttamiseen käytetyn ajan arvoista.

Tämänkaltaisessa tapauksessa, jossa sivulle päivitetään sisältöä vain yhteen kohtaan, voisi olla nopeampi ja vaivattomampi tapa laittaa palvelin palauttamaan vastaus, tässä tapauksessa siis uusi viesti, valmiiksi XHTML-muotoiltuna XML:n sijaan. Tällöin säästyttäisiin datan muotoilulta JavaScript-funktiossa, kun valmis pala XHTML:ää voitaisiin upottaa dokumenttiin sellaisenaan. Dataformaatin valinta tulee siis tehdä aina tapauskohtaisesti.

Yleisesti ottaen Ajax-toimintojen suunnitteluun on syytä paneutua huolella, toiminto kerrallaan. Näin voidaan säästää huomattavasti aikaa ja vaivaa, kun niitä toteutetaan ja implementoidaan valmiin käyttöliittymän päälle.

8 Yhteenveto

Ajax on todistanut olevansa tähän asti käyttökelpoisin, ilman selainlisäosia toimiva tekniikka selaimen ja palvelimen välisen asynkronisen kommunikaation toteuttamiseen. Se on vastaus moniin muiden ratkaisujen leviämistä osaltaan hidastaneisiin yhteensopivuusongelmiin, joita useiden eri versionumerolla varustettujen selainlisäosien kanssa on jouduttu kokemaan, sillä se hyödyntää lähes kaikissa moderneissa selaimissa natiivina toteutettuja komponentteja ja laajalti tuettuja, avoimiin standardeihin perustuvia teknologioita ja dataformaatteja. Se on yksi vastaus myös käyttäjien tarpeisiin rikkaammista verkkosovelluksista ja samalla ratkaisu kehittäjien ongelmaan ohjelmistopäivitysten jakelusta.

Koska Ajax hyödyntää vanhoja teknologioita, on sen oppimiskynnys erityisesti kokeneemmalle web-kehittäjälle matala – eikä vähiten siitä syystä, että Ajax ei aseta vaatimuksia palvelimella käytettävän ohjelmointikielen suhteen. Olemassa olevien valmiiden työkalujen, kuten JavaScript-kirjastojen, avulla jokaisen kehittäjän on siis mahdollista luoda omia tarpeitaan vastaava paletti Ajax-kehitykseen suhteellisen vaivattomasti. Matalan oppimiskynnyksen vuoksi Ajax on yleistynyt jo vain muutaman vuoden aikana monien kehittäjien päivittäin hyödyntämäksi tekniikaksi, mikä näkyy internetissä mielenkiintoisena valikoimana toinen toistaan kekseliäämpiä sovelluksia, ja mikä parasta, yleisellä tasolla sivustojen käytettävyyden parantumisena.

Aloitin itse tutustumisen Ajaxiin noin kaksi vuotta Garrettin artikkelin, jossa hän määritteli termin ensimmäistä kertaa, julkaisun jälkeen. Vielä työni kirjoitusprosessin aikana olen kokenut oppineeni Ajaxista ja tavoista soveltaa sitä vähintään yhtä paljon, ellen jopa enemmän, kuin kaksi vuotta sitten tekniikkaan ensimmäistä kertaa tietoisesti törmätessäni. Käsitykseni Ajaxista ja sen mahdollisuuksista ovat siis kirkastuneet jatkuvasti, joten päätin rajata aiheen käsittelyn yleisluontoiseksi johdannoksi Ajaxin perusteisiin ja jättää syvempää teknistä osaamista vaativia asioita rajauksen ulkopuolelle. Tekstissäni käsittelemät aiheet tarjoavat vahvan pohjan, josta lähteä kartoittamaan Ajaxin mahdollisuuksia pidemmälle.

Ajaxin ympärillä on ollut niin jatkuvaa hypeä kuin kriittistä keskusteluakin koko sen tähänastisen elinkaaren ajan. Sen on epäilty olevan vain yksi monista ohimenevistä

villityksistä, mikä kieltämättä on luonnollinen vastareaktio suosioon noussutta, uutta tekniikkaa kohtaan. Tällä hetkellä, useiden merkittävien alan toimijoiden, kuten Googlen, Amazonin ja Microsoftin, aktiivisesti kehittäen uusia sovelluksia sitä hyödyntäen kuitenkin näyttäisi vahvasti siltä, että Ajax on tullut jäädäkseen.

Lähteet

Asleson, Ryan & Schutta, Nathaniel T. 2006. *Ajax – Tehokas hallinta.* Helsinki, Suomi: Readme.fi.

Crockford, Douglas. 2006. *JSON – The Fat-Free Alternative to XML.* Artikkeli, http://www.json.org/fatfree.html (Luettu 18.11.2008)

Edwards, James. 2006. *AJAX and Screenreaders: When Can it Work?.* Artikkeli, http://www.sitepoint.com/article/ajax-screenreaders-work/ (Luettu 3.11.2008)

Garrett, Jesse James. 18.2.2005. *Ajax: A New Approach to Web Applications.* Artikkeli, http://www.adaptivepath.com/ideas/essays/archives/000385.php (Luettu 2.10.2008)

Holdener III, Anthony T. 2008. *Ajax: The Definitive Guide.* Sebastopol, CA, USA: O'Reilly Media, Inc.

Internet World Stats. *Internet Usage Statistics: The Internet Big Picture, World Internet Users and Population Stats.* Tilasto, http://www.internetworldstats.com/stats.htm (Luettu, 5.11.2008)

Keith, Jeremy. 2007. *Bulletproof Ajax.* Berkeley, CA, USA: New Riders.

Mozilla Developer Center. *Core JavaScript 1.5 Reference: eval.* http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Global_Functions:eval (Luettu 18.11.2008)

O'Reilly, Tim, 30.9.2005. *What is Web 2.0 – Design patterns and business models for the next generation of software.* Artikkeli, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html (Luettu 20.11.2008)

Wikipedia. *Progressive enhancement.* Artikkeli, http://en.wikipedia.org/wiki/Progressive_enhancement (Luettu, 4.11.2008)

Wikipedia. *XML.* Artikkeli, http://en.wikipedia.org/wiki/XML (Luettu, 23.11.2008)

W3C. *The XMLHttpRequest Object: W3C Working Draft 15 April 2008.* http://www.w3.org/TR/2008/WD-XMLHttpRequest-20080415/ (Luettu 18.10.2008)

W3Schools. *The XMLHttpRequest Object.* http://www.w3schools.com/dom/dom_http.asp (Luettu 18.10.2008)